CS 5220: Homework 2

Group 19: Robert Carson (rac428), Robert Chiodi (rmc298), Sam Tung (sat83)

October 14, 2015

1 Initial Profiling

In order to perform initial profiling of the code before any improvements are made, Intel's VTUNE was used via the terminal command line on Totient. In an attempt to get the most accurate results (taken with a large sample size), we decided to gather data while running the large wave simulation, invoked with the command `make big`. Information collected from VTUNE's 'advanced-hotspots' option is used in our analysis.

1.1 Whole Program - Advanced Hotspots

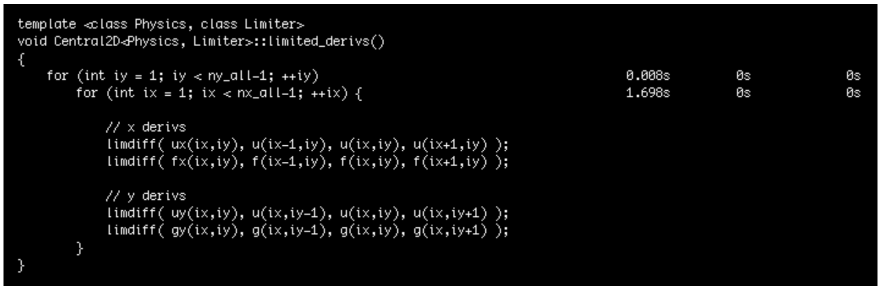
First, hotspots in the entire program were examined in order to determine where our efforts should be directed. The time taken in the top 10 most time consuming functions can be seen below in Figure 1. Of these, it is clear that most of our optimization efforts should be directed to the functions `limited_derivs`, `compute_step`, and `compute_fg_speeds`. These functions were then examined individually, once again using VTUNE on our advanced-hotspots collection.

```
amplxe: Executing actions 50 % Generating a report                         Function
   Module        CPU Time  Spin Time  Overhead Time
-------------------------------------------------------------------------------------------------  -----------  --------  ---------  ------------
Central2D<Shallow2D, MinMod<float>>::limited_derivs                                                 shallow      279.792s        0s            0s
Central2D<Shallow2D, MinMod<float>>::compute_step                                                   shallow      150.034s        0s            0s
Central2D<Shallow2D, MinMod<float>>::compute_fg_speeds                                              shallow       49.277s        0s            0s
[Outside any known module]                                                                          [Unknown]      1.479s        0s            0s
_IO_file_xsputn                                                                                     libc-2.12.so   0.700s        0s            0s
_IO_fwrite                                                                                          libc-2.12.so   0.679s        0s            0s
Central2D<Shallow2D, MinMod<float>>::run                                                            shallow        0.307s        0s            0s
SimViz<Central2D<Shallow2D, MinMod<float>>>::write_frame                                            shallow        0.242s        0s            0s
Central2D<Shallow2D, MinMod<float>>::solution_check                                                 shallow        0.171s        0s            0s
std::vector<std::array<float, (unsigned long)3>, std::allocator<std::array<float, (unsigned long)3>>>::operator[] shallow        0.044s        0s            0s
```

Figure 1: Top 10 most time consuming functions in the wave simulation. Generated using Intel's VTUNE on Totient.

1.2 `limited_derivs` - Advanced Hotspots

The function `limited_deriv` is used to calculate the fluxes into and out of each cell in order to advance to the next time step. This involves a three point computational stencil in each direction and loops through the entire domain interior (the whole domain except for those where boundary conditions are applied). Each point requires `du.size()` $\times$ 9 floating point operations as well as `du.size()` $\times$ 2 calls to the intrinsic function `min`. Sadly, the hotspot analysis on the `limited_deriv` function, shown in Figure 2, does not give any hints on possible optimizations or bottle necks.



```
template <class Physics, class Limiter>
void Central2D<Physics, Limiter>::limited_derivs()
{
    for (int iy = 1; iy < ny_all-1; ++iy)                              0.008s     0s     0s
        for (int ix = 1; ix < nx_all-1; ++ix) {                        1.698s     0s     0s

            // x derivs
            limdiff( ux(ix,iy), u(ix-1,iy), u(ix,iy), u(ix+1,iy) );
            limdiff( fx(ix,iy), f(ix-1,iy), f(ix,iy), f(ix+1,iy) );

            // y derivs
            limdiff( uy(ix,iy), u(ix,iy-1), u(ix,iy), u(ix,iy+1) );
            limdiff( gy(ix,iy), g(ix,iy-1), g(ix,iy), g(ix,iy+1) );
        }
}
```

Figure 2: Time taken to perform each loop present in `limited_derivs`

1.3 `compute_step` - Advanced Hotspots

The purpose of `compute_step` is to update the wave equation to the next time step using a predictor-corrector method. First, the fluxes are calculated in the prediction. Next, the corrector step uses the predicted fluxes, the differences in velocities, and the current velocities to advance to the next time state. Luckily, VTUNE's report is more helpful than in the previous case, and provides extensive timings for this function, shown in Figure 3. The calculation in the corrector step can be seen to be the most expensive cost of the function. It is important to note, however, that the predictor step and copying of the solution to the `u` array sum to half of the function's cost.

1.4 `compute_fg_speeds` - Advanced Hotspots

The function of `compute_fg_speeds` has two primary responsibilities: to update the cell centered fluxes, `f` and `g`, and to calculate the maximum speed in the domain, allowing dynamic adjustment of the time step in order to satisfy the CFL condition and ensure numerical stability. The timing data for this function can be seen in Figure 4. While the most time consuming portion of the code is most likely the calculation of the fluxes and wave speed (both of which are in the `Shallow2d` structure), the calls to the intrinsic function `max` also represent a non-trivial amount of time.

```
template <class Physics, class Limiter>
void Central2D<Physics, Limiter>::compute_step(int io, real dt)
{
    real dtcdx2 = 0.5 * dt / dx;
    real dtcdy2 = 0.5 * dt / dy;

    // Predictor (flux values of f and g at half step)
    for (int iy = 1; iy < ny_all-1; ++iy)                                0.003s    0s    0s
        for (int ix = 1; ix < nx_all-1; ++ix) {                          0.357s    0s    0s
            vec uh = u(ix,iy);                                           1.909s    0s    0s
            for (int m = 0; m < uh.size(); ++m) {
                uh[m] -= dtcdx2 * fx(ix,iy)[m];
                uh[m] -= dtcdy2 * gy(ix,iy)[m];                          7.334s    0s    0s
            }
            Physics::flux(f(ix,iy), g(ix,iy), uh);
        }

    // Corrector (finish the step)
    for (int iy = nghost-io; iy < ny+nghost-io; ++iy)                    0.010s    0s    0s
        for (int ix = nghost-io; ix < nx+nghost-io; ++ix) {              1.750s    0s    0s
            for (int m = 0; m < v(ix,iy).size(); ++m) {
                v(ix,iy)[m] =                                           26.781s    0s    0s
                    0.2500 * ( u(ix,  iy)[m] + u(ix+1,iy  )[m] +         1.553s    0s    0s
                               u(ix,iy+1)[m] + u(ix+1,iy+1)[m] ) -       6.718s    0s    0s
                    0.0625 * ( ux(ix+1,iy  )[m] - ux(ix,iy  )[m] +
                               ux(ix+1,iy+1)[m] - ux(ix,iy+1)[m] +
                               uy(ix,  iy+1)[m] - uy(ix,  iy)[m] +
                               uy(ix+1,iy+1)[m] - uy(ix+1,iy)[m] ) -     1.501s    0s    0s
                    dtcdx2 * ( f(ix+1,iy  )[m] - f(ix,iy  )[m] +         0.228s    0s    0s
                               f(ix+1,iy+1)[m] - f(ix,iy+1)[m] ) -       7.972s    0s    0s
                    dtcdy2 * ( g(ix,  iy+1)[m] - g(ix,  iy)[m] +         1.840s    0s    0s
                               g(ix+1,iy+1)[m] - g(ix+1,iy)[m] );        8.207s    0s    0s
            }
        }

    // Copy from v storage back to main grid
    for (int j = nghost; j < ny+nghost; ++j){                            0.003s    0s    0s
        for (int i = nghost; i < nx+nghost; ++i){                        1.624s    0s    0s
            u(i,j) = v(i-io,j-io);                                      10.566s    0s    0s
        }
    }
}                                                                        0.001s    0s    0s
```

Figure 3: Time taken to perform each loop present in `limited_derivs`

```
{
    using namespace std;
    real cx = 1.0e-15;
    real cy = 1.0e-15;
    for (int iy = 0; iy < ny_all; ++iy)                          0.001s    0s    0s
        for (int ix = 0; ix < nx_all; ++ix) {                    3.240s    0s    0s
            real cell_cx, cell_cy;
            Physics::flux(f(ix,iy), g(ix,iy), u(ix,iy));
            Physics::wave_speed(cell_cx, cell_cy, u(ix,iy));
            cx = max(cx, cell_cx);                               4.299s    0s    0s
            cy = max(cy, cell_cy);                               1.666s    0s    0s
        }
    cx_ = cx;
    cy_ = cy;
}
```

Figure 4: Time taken to perform each loop present in `limited_derivs`

References